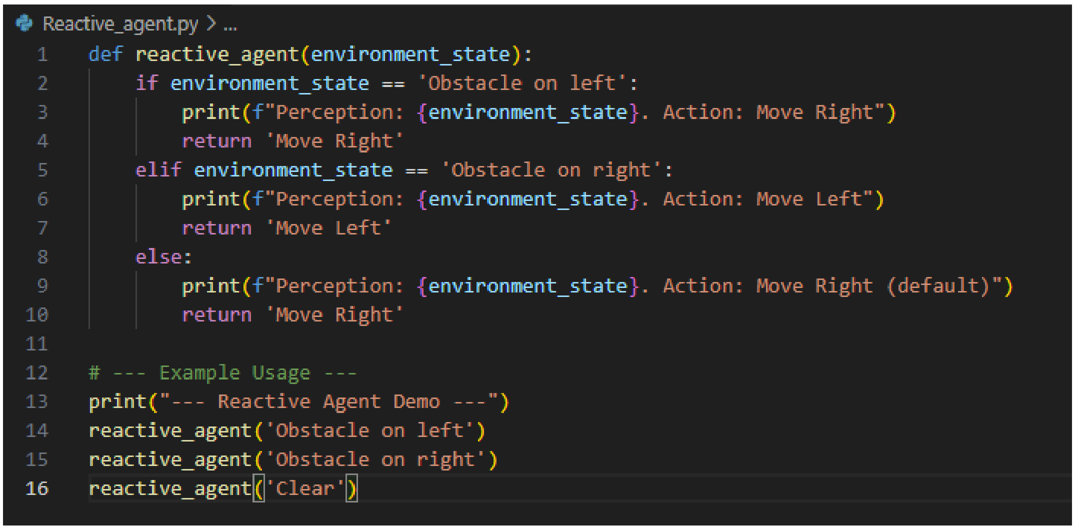
AIR UNIVERSITY

Agent Based Software Engineering Lab

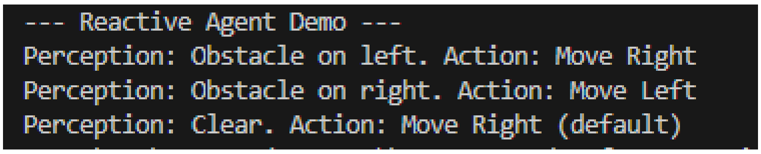
LabTask 02

Name : Muhammad Shamoil
Section : BSSE – V – A
Date: 16th October , 2025
Submitted To : Sir Wasim Wahid

1. Reactive Agent

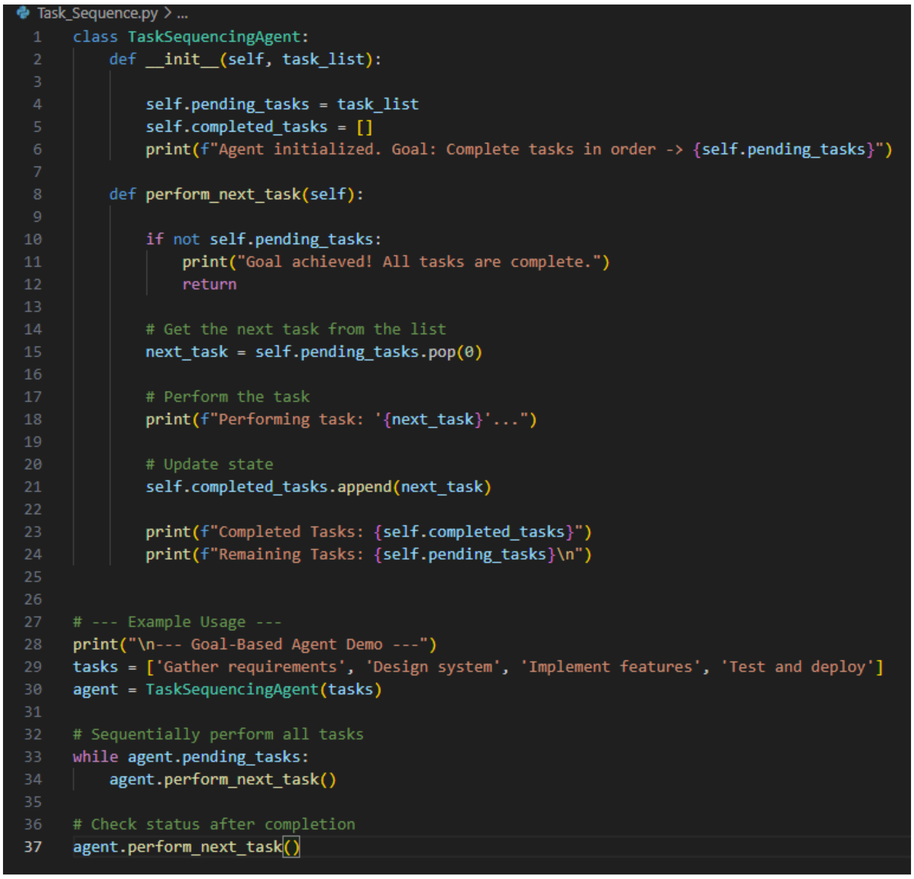


```python
def reactive_agent(environment_state):
    if environment_state == 'Obstacle on left':
        print(f"Perception: {environment_state}. Action: Move Right")
        return 'Move Right'
    elif environment_state == 'Obstacle on right':
        print(f"Perception: {environment_state}. Action: Move Left")
        return 'Move Left'
    else:
        print(f"Perception: {environment_state}. Action: Move Right (default)")
        return 'Move Right'

# --- Example Usage ---
print("--- Reactive Agent Demo ---")
reactive_agent('Obstacle on left')
reactive_agent('Obstacle on right')
reactive_agent('Clear')
```

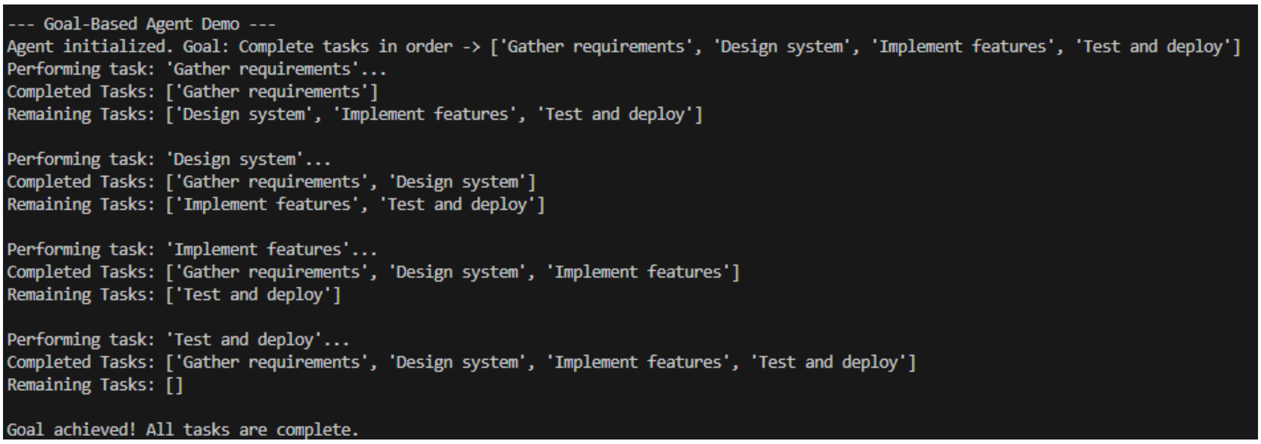


```
--- Reactive Agent Demo ---
Perception: Obstacle on left. Action: Move Right
Perception: Obstacle on right. Action: Move Left
Perception: Clear. Action: Move Right (default)
```

2. Task Sequencing Agent

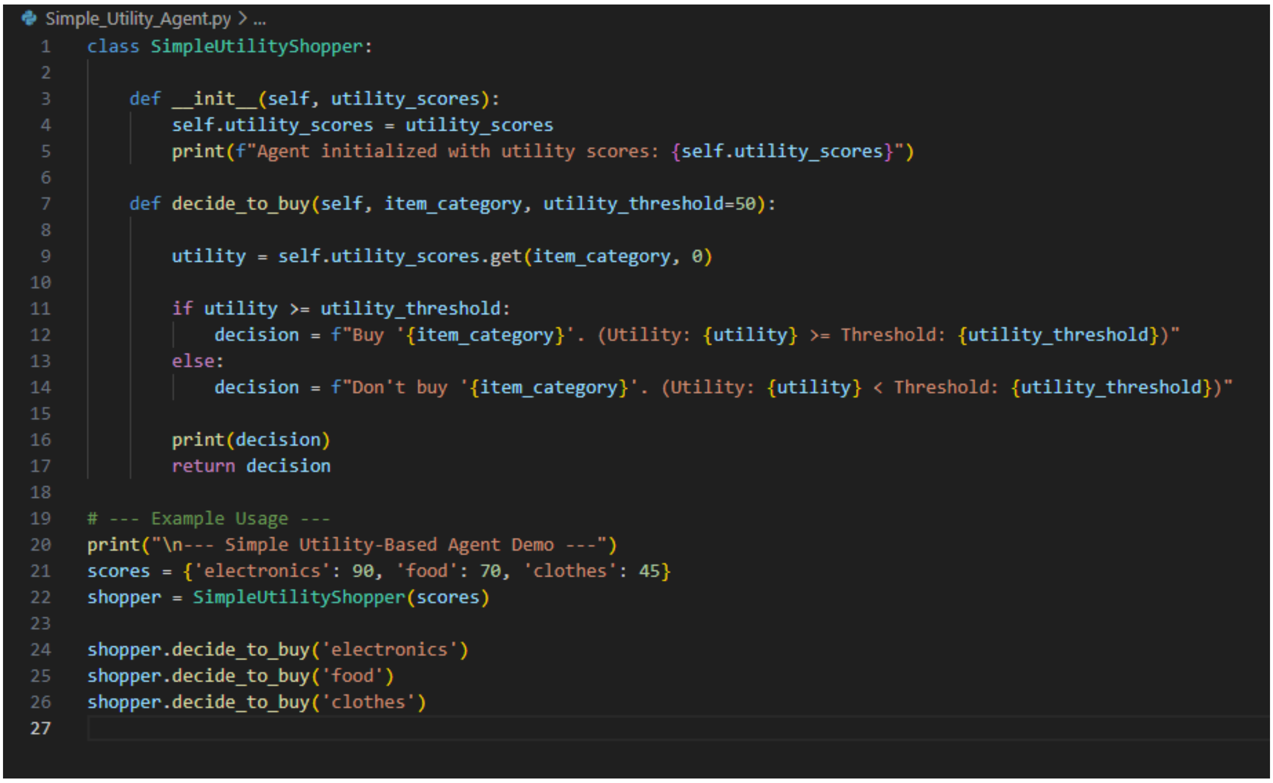


```python
class TaskSequencingAgent:
    def __init__(self, task_list):

        self.pending_tasks = task_list
        self.completed_tasks = []
        print(f"Agent initialized. Goal: Complete tasks in order -> {self.pending_tasks}")

    def perform_next_task(self):

        if not self.pending_tasks:
            print("Goal achieved! All tasks are complete.")
            return

        # Get the next task from the list
        next_task = self.pending_tasks.pop(0)

        # Perform the task
        print(f"Performing task: '{next_task}'...")

        # Update state
        self.completed_tasks.append(next_task)

        print(f"Completed Tasks: {self.completed_tasks}")
        print(f"Remaining Tasks: {self.pending_tasks}\n")


# --- Example Usage ---
print("\n--- Goal-Based Agent Demo ---")
tasks = ['Gather requirements', 'Design system', 'Implement features', 'Test and deploy']
agent = TaskSequencingAgent(tasks)

# Sequentially perform all tasks
while agent.pending_tasks:
    agent.perform_next_task()

# Check status after completion
agent.perform_next_task()
```

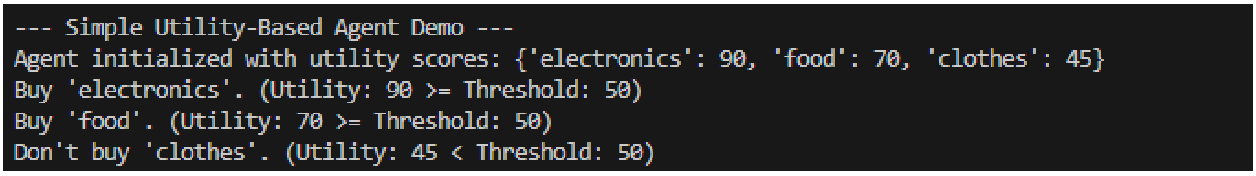


```
--- Goal-Based Agent Demo ---
Agent initialized. Goal: Complete tasks in order -> ['Gather requirements', 'Design system', 'Implement features', 'Test and deploy']
Performing task: 'Gather requirements'...
Completed Tasks: ['Gather requirements']
Remaining Tasks: ['Design system', 'Implement features', 'Test and deploy']

Performing task: 'Design system'...
Completed Tasks: ['Gather requirements', 'Design system']
Remaining Tasks: ['Implement features', 'Test and deploy']

Performing task: 'Implement features'...
Completed Tasks: ['Gather requirements', 'Design system', 'Implement features']
Remaining Tasks: ['Test and deploy']

Performing task: 'Test and deploy'...
Completed Tasks: ['Gather requirements', 'Design system', 'Implement features', 'Test and deploy']
Remaining Tasks: []

Goal achieved! All tasks are complete.
```

3. Simple Utility Shopper

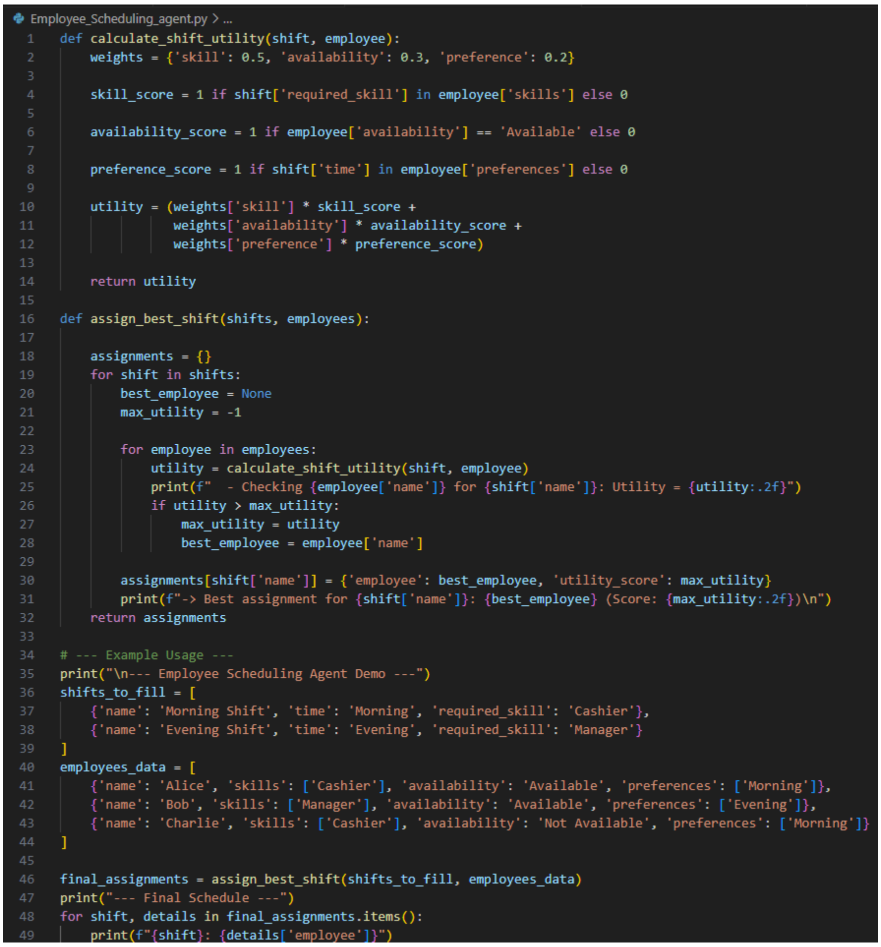


```python
class SimpleUtilityShopper:

    def __init__(self, utility_scores):
        self.utility_scores = utility_scores
        print(f"Agent initialized with utility scores: {self.utility_scores}")

    def decide_to_buy(self, item_category, utility_threshold=50):

        utility = self.utility_scores.get(item_category, 0)

        if utility >= utility_threshold:
            decision = f"Buy '{item_category}'. (Utility: {utility} >= Threshold: {utility_threshold})"
        else:
            decision = f"Don't buy '{item_category}'. (Utility: {utility} < Threshold: {utility_threshold})"

        print(decision)
        return decision

# --- Example Usage ---
print("\n--- Simple Utility-Based Agent Demo ---")
scores = {'electronics': 90, 'food': 70, 'clothes': 45}
shopper = SimpleUtilityShopper(scores)

shopper.decide_to_buy('electronics')
shopper.decide_to_buy('food')
shopper.decide_to_buy('clothes')

```



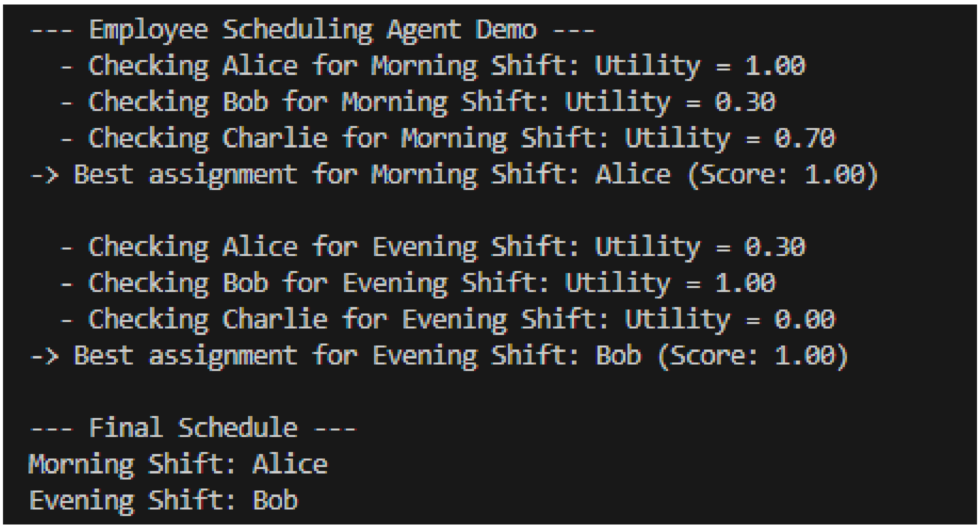
```
--- Simple Utility-Based Agent Demo ---
Agent initialized with utility scores: {'electronics': 90, 'food': 70, 'clothes': 45}
Buy 'electronics'. (Utility: 90 >= Threshold: 50)
Buy 'food'. (Utility: 70 >= Threshold: 50)
Don't buy 'clothes'. (Utility: 45 < Threshold: 50)
```

4. Calculate Shift Utility



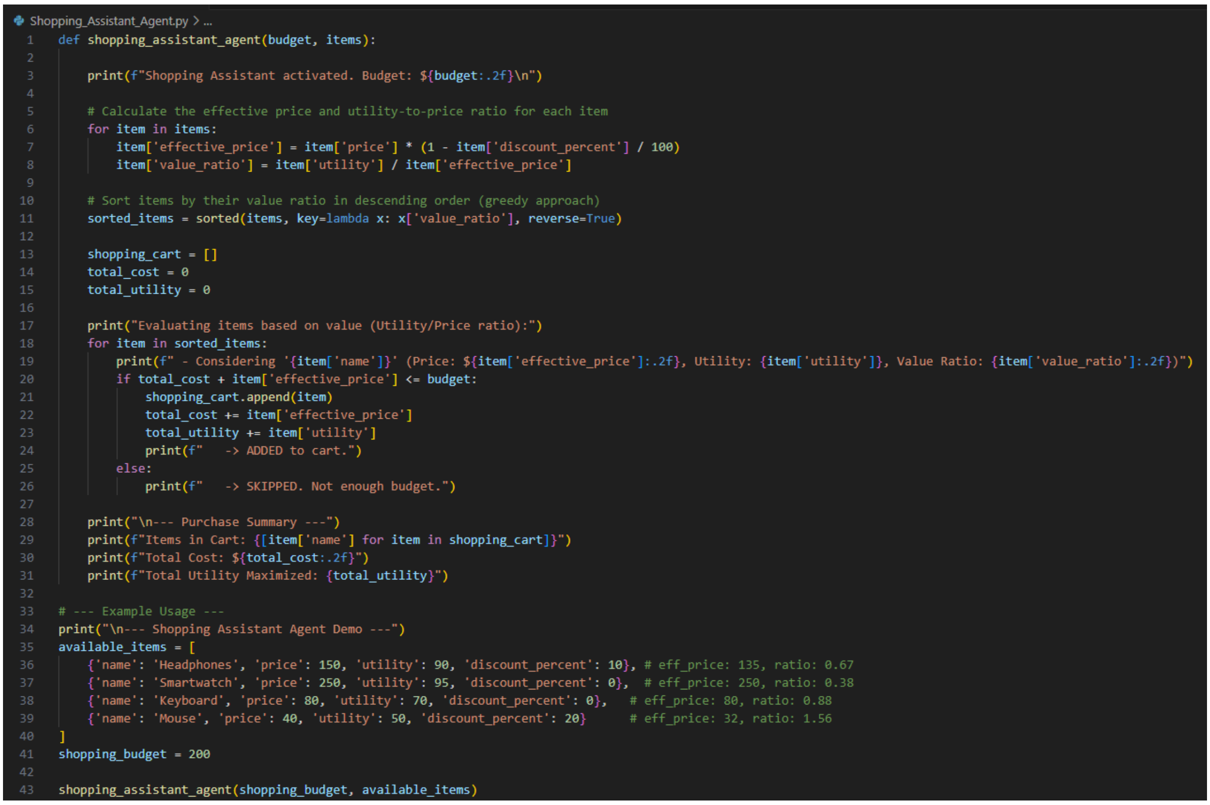
Employee_Scheduling_agent.py > ...

```python
def calculate_shift_utility(shift, employee):
    weights = {'skill': 0.5, 'availability': 0.3, 'preference': 0.2}

    skill_score = 1 if shift['required_skill'] in employee['skills'] else 0

    availability_score = 1 if employee['availability'] == 'Available' else 0

    preference_score = 1 if shift['time'] in employee['preferences'] else 0

    utility = (weights['skill'] * skill_score +
               weights['availability'] * availability_score +
               weights['preference'] * preference_score)

    return utility

def assign_best_shift(shifts, employees):

    assignments = {}
    for shift in shifts:
        best_employee = None
        max_utility = -1

        for employee in employees:
            utility = calculate_shift_utility(shift, employee)
            print(f"  - Checking {employee['name']} for {shift['name']}: Utility = {utility:.2f}")
            if utility > max_utility:
                max_utility = utility
                best_employee = employee['name']

        assignments[shift['name']] = {'employee': best_employee, 'utility_score': max_utility}
        print(f"-> Best assignment for {shift['name']}: {best_employee} (Score: {max_utility:.2f})\n")
    return assignments

# --- Example Usage ---
print("\n--- Employee Scheduling Agent Demo ---")
shifts_to_fill = [
    {'name': 'Morning Shift', 'time': 'Morning', 'required_skill': 'Cashier'},
    {'name': 'Evening Shift', 'time': 'Evening', 'required_skill': 'Manager'}
]
employees_data = [
    {'name': 'Alice', 'skills': ['Cashier'], 'availability': 'Available', 'preferences': ['Morning']},
    {'name': 'Bob', 'skills': ['Manager'], 'availability': 'Available', 'preferences': ['Evening']},
    {'name': 'Charlie', 'skills': ['Cashier'], 'availability': 'Not Available', 'preferences': ['Morning']}
]

final_assignments = assign_best_shift(shifts_to_fill, employees_data)
print("--- Final Schedule ---")
for shift, details in final_assignments.items():
    print(f"{shift}: {details['employee']}")
```



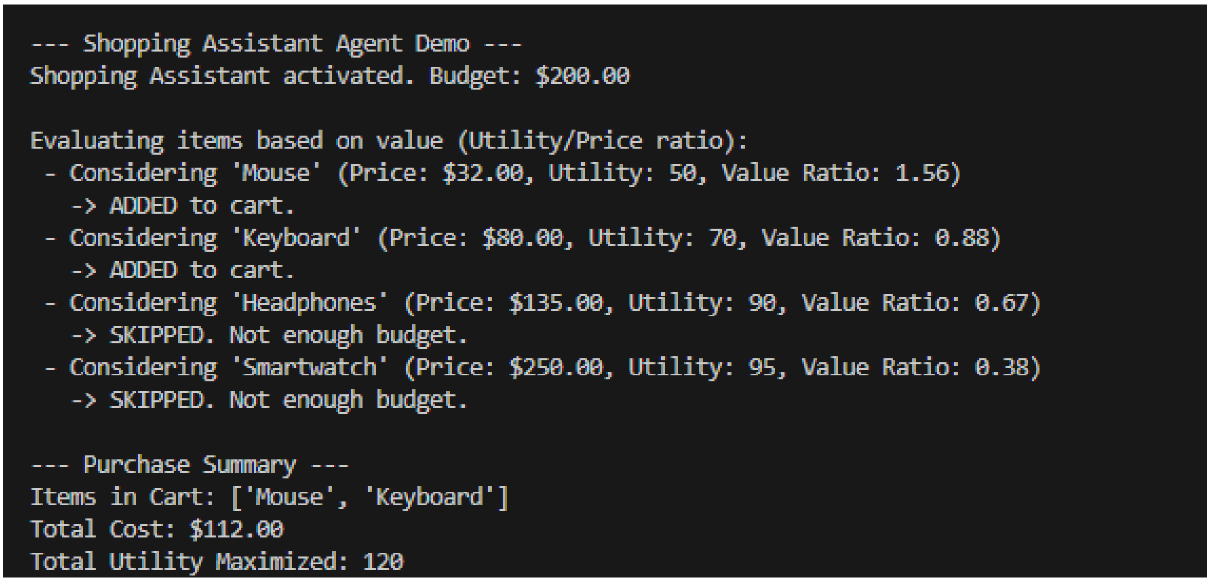
```
--- Employee Scheduling Agent Demo ---
  - Checking Alice for Morning Shift: Utility = 1.00
  - Checking Bob for Morning Shift: Utility = 0.30
  - Checking Charlie for Morning Shift: Utility = 0.70
-> Best assignment for Morning Shift: Alice (Score: 1.00)

  - Checking Alice for Evening Shift: Utility = 0.30
  - Checking Bob for Evening Shift: Utility = 1.00
  - Checking Charlie for Evening Shift: Utility = 0.00
-> Best assignment for Evening Shift: Bob (Score: 1.00)

--- Final Schedule ---
Morning Shift: Alice
Evening Shift: Bob
```

5. Shopping Assistant Agent



Shopping_Assistant_Agent.py > ...

```python
def shopping_assistant_agent(budget, items):

    print(f"Shopping Assistant activated. Budget: ${budget:.2f}\n")

    # Calculate the effective price and utility-to-price ratio for each item
    for item in items:
        item['effective_price'] = item['price'] * (1 - item['discount_percent'] / 100)
        item['value_ratio'] = item['utility'] / item['effective_price']

    # Sort items by their value ratio in descending order (greedy approach)
    sorted_items = sorted(items, key=lambda x: x['value_ratio'], reverse=True)

    shopping_cart = []
    total_cost = 0
    total_utility = 0

    print("Evaluating items based on value (Utility/Price ratio):")
    for item in sorted_items:
        print(f" - Considering '{item['name']}' (Price: ${item['effective_price']:.2f}, Utility: {item['utility']}, Value Ratio: {item['value_ratio']:.2f})")
        if total_cost + item['effective_price'] <= budget:
            shopping_cart.append(item)
            total_cost += item['effective_price']
            total_utility += item['utility']
            print(f"   -> ADDED to cart.")
        else:
            print(f"   -> SKIPPED. Not enough budget.")

    print("\n--- Purchase Summary ---")
    print(f"Items in Cart: {[item['name'] for item in shopping_cart]}")
    print(f"Total Cost: ${total_cost:.2f}")
    print(f"Total Utility Maximized: {total_utility}")

# --- Example Usage ---
print("\n--- Shopping Assistant Agent Demo ---")
available_items = [
    {'name': 'Headphones', 'price': 150, 'utility': 90, 'discount_percent': 10}, # eff_price: 135, ratio: 0.67
    {'name': 'Smartwatch', 'price': 250, 'utility': 95, 'discount_percent': 0},  # eff_price: 250, ratio: 0.38
    {'name': 'Keyboard', 'price': 80, 'utility': 70, 'discount_percent': 0},   # eff_price: 80, ratio: 0.88
    {'name': 'Mouse', 'price': 40, 'utility': 50, 'discount_percent': 20}      # eff_price: 32, ratio: 1.56
]
shopping_budget = 200

shopping_assistant_agent(shopping_budget, available_items)
```



```
--- Shopping Assistant Agent Demo ---
Shopping Assistant activated. Budget: $200.00

Evaluating items based on value (Utility/Price ratio):
 - Considering 'Mouse' (Price: $32.00, Utility: 50, Value Ratio: 1.56)
   -> ADDED to cart.
 - Considering 'Keyboard' (Price: $80.00, Utility: 70, Value Ratio: 0.88)
   -> ADDED to cart.
 - Considering 'Headphones' (Price: $135.00, Utility: 90, Value Ratio: 0.67)
   -> SKIPPED. Not enough budget.
 - Considering 'Smartwatch' (Price: $250.00, Utility: 95, Value Ratio: 0.38)
   -> SKIPPED. Not enough budget.

--- Purchase Summary ---
Items in Cart: ['Mouse', 'Keyboard']
Total Cost: $112.00
Total Utility Maximized: 120
```